

Using the Java Language to Develop Computer Based Patient Records for use on the Internet

Alan E. Zuckerman, M.D.

Departments of Family Medicine and Pediatrics
Georgetown University School of Medicine
Washington, DC

The development of the Java Programming Language by Sun Microsystems has provided a new tool for the development of Internet based applications. Our preliminary work has shown how Java can be used to program an Internet based CBPR. Java is well suited to the needs of patient records and can interface with clinical data repositories written in MUMPS or SQL.

INTRODUCTION

While most authors have judged that technical barriers play a small role in constraining the growth of Computer-based Patient Records (CBPR),¹ the dissemination of CBPR across the Internet demands more secure and robust technology than that used by a standalone CBPR. Using the World-Wide Web as a distributed CBPR has been shown to be feasible but may not be an optimal approach.² New operating systems and new computer platforms emphasize the need for Platform Independent and Portable CBPR systems.

This paper will discuss the design goals of the Java language³ developed by Sun Microsystems and adopted by many other vendors and their relevance to the needs of a CBPR. The functional requirements of a CBPR and how they will can be met using Java will be reviewed along with alternative strategies for implementing Internet based CBPR.

Java has much in Common with MUMPS

The MUMPS programming language was developed by Greenes and colleagues 25 years ago⁵ and had a profound effect on the development of Medical Informatics and the CBPR. The Computer Stored Ambulatory Record (COSTAR) remains in widespread use and was one of the first applications developed using ANSI Standard MUMPS. Like MUMPS, Java is Interpreted and designed to make systems and programs more responsive to the demands of the clinical environment than alternative systems of its day. Like MUMPS, Java is relatively simple and easy to learn allowing a single individual to deploy complex system functions without the need to learn and program details of the operating system implementation. Both languages have earned praise as rapid prototyping tools, but unlike MUMPS, Java is not criticized for being too powerful to be safe, or

too cryptic for one programmer to understand what another has done.

DESIGN GOALS OF JAVA

Java is Multi-platform

The most striking new feature of Java is the ability to run binary Java Bytecodes (the instruction set of the Java Virtual Machine) on a variety of computers and operating systems with the look and feel of native applications. Developing a CBPR in Java allows the same program code to run on Windows, Unix and MacOS. The Java Advanced Windows Toolkit (AWT) provides a common Graphic User Interface (GUI) which is easily moved across platforms. Part of the AWT are intelligent layout managers which place the actual interface components within their containers. The Java Virtual Machine is being implemented in custom microprocessors and Sun is developing a simple operating system to allow building inexpensive Java computers for Internet access. The performance of Java can be improved through the use of Just-in-Time compilers, such as the one recently added to Netscape Navigator 3.0, which translate the original Bytecode output of the Java compiler into specific machine code while the program is being read from the disk or network.

Java is Object-Oriented

Java was based on C++ with several features removed, and a few added to force programmers to adhere to a strict object-oriented paradigm. In Java strings are treated as objects rather than arrays. A strictly object-oriented language has advantages for developing a CBPR by facilitating sharing of reusable modules of code and enabling local extensions and variations to build on a common system structure.

Java is Dynamic

The Java language was designed to adapt to changing environments found on networks and to facilitate the development of self maintaining code. The HotJava web browser was the first substantial application written entirely in Java and was designed as a modular self-updating application designed to integrate new protocols and new data types in a seamless dynamic fashion. A dynamic and extensible language is useful for developing a CBPR

Java is Safe and Secure

Java was designed to address network security and prevention of programming errors which are two fundamental problems in modern software development. The Internet is inherently public and insecure, therefore Java has safeguards to prevent abuse. Virus resistance was designed into the Java language. While no system can be perfect, the design goals have received considerable attention and detected failures of the network security have been corrected rapidly. Java Applets which are loaded from web pages are constrained to make them secure by restricting file access, TCP/IP sockets, and URLs to the server where they originated. Java applications installed on the user's workstation can function over the Internet in a client server mode as well as access local disk files. The Java language was designed to prevent programming bugs and errors by forcing programmers to anticipate error conditions or their programs will not compile. Java is a strongly typed language which has also removed many potentially error prone features of C++ such as pointer arithmetic and memory allocation.

Java is Multi-Threaded

Multi-threaded programming was designed into the Java language and does not require knowledge or use of any special operating system features. Individual threads run concurrently as part of the same application process and are synchronized when necessary by Java monitors. Java implements its own priority and scheduling algorithms to control threads. Multi-threaded operations are useful for a CBPR to allow automated quality assurance to occur in the background with immediate feedback to the physician. They can also be used to support assembling an integrated patient record from multiple practices distributed over the Internet. Threads can be used to insert random and spontaneous behavior into a user interaction increasing user alertness and helping users to learn complex systems and browse complex data.

Java is Network Aware

Java was designed to run in a network environment and provides the programmer with easy access to all basic network protocols and functions which facilitates and speeds developing client server applications. Data can be distributed and files or images are located using standard Uniform Resource Locator (URL) syntax. The network capabilities of Java can support a distributed CBPR spanning multiple sites and providers as well as support distributed computing within a single institution.

Java Uses the JDBC to Access SQL Servers

The Java Database Connectivity (JDBC) Application Programming Interface (API) allows Java

programmers to access SQL databases in a standardized, platform independent, and vendor independent mode.⁴ JDBC was based on the Open Database Connectivity (ODBC) model and JDBC provides a bridge for using existing ODBC drivers until vendors provide their own JDBC drivers. The JDBC drivers require no operating system configuration on each client because all environment settings are made using JDBC calls.

Java is Not Stateless

A common misconception about Java is the assumption that since it is part of the World Wide Web (WWW) it suffers from the limitations of statelessness which must be overcome by other strategies for implementing CBPR access through web browsers. Java is a programming language and has the same attributes of state found in any C language program. Java can take advantage of stateless connections to any URL on the WWW, but it also can maintain its own dedicated TCP/IP sockets or use local or remote disk file access to maintain state.

Java is Being Represented as a Paradigm Shift

The developers of Java are trying to promote it as more than a new programming language, but instead as a key enabling technology in a paradigm shift as significant in the history of computing as the development of stored programs, the shift from mainframes to minicomputers, and the shift from minicomputers to personal computers. As in previous paradigm shifts, the old technology does not disappear, thus we still have mainframes, but we now have choices. Even in the Network Age, we will still use platform specific complex applications written in C++, but we will have the choice of platform independent, network aware, Java applications. While development and use of minicomputer based CBPR will continue, the Java paradigm offers attractive choices.

SOFTWARE REQUIREMENTS OF CBPR

Shortliffe, Barnett, and McDonald⁶ have summarized the key functional requirements of a CBPR which form a framework for evaluating Java.

Database Structure

The Database structure of a CBPR constrains the type of data elements, their relationships, and indexing within a system. The proposed ASTM standard for structure of the CBPR defines seven key linked components of structure: Patient, Provider, Encounter, Problem, Observation, Order, and Service. Java is well suited to implementing the type of data structure required by a CBPR. Java classes provide an effective tool for implementing an object

oriented database model for a CBPR using standard SQL tools.

Content

The content of a CBPR refers to the range of clinical data it includes. A mini-record might consist only of a problem list, medication list, and lab summary while a complete record would include many sections including allergies, immunizations, and radiology. Complete records will incorporate images and free-text as well as coded data and numeric results. Java provides methods for coping with the variable content of different CBPRs by writing Java classes as "Content Handlers" for different types of content in a CBPR. Appropriate content handling classes are downloaded and used dynamically to handle new content. The classes for handling the content are stored on the same server with the data itself. Java can also customize display menus and format in response to a particular implementation of a CBPR.

Coding

The utility and uniformity of a CBPR is highly dependent upon the ability to code as much of the information in the record as possible. Diagnoses, Medications, and Procedures have widely accepted coding schemes, but coding of the rest of the clinical encounter remains an area of active development and study. The Uniform Medical Language System (UMLS)⁷ project of the National Library of Medicine (NLM) is helping to promote sharing and translation of clinical coding systems. Java is well suited to integrating UMLS databases and concepts into a CBPR through reusable software modules implemented as Java classes. Java also provides a mechanism for sharing local coding systems on an as needed basis through client server applications.

An integrated lifelong CBPR creates additional coding challenges due to the variations in level of detail used by different specialties and physicians for the same types of examinations. Consider the coding of the physical examination of the ear done by a Pediatrician, an Internist, or an Otolaryngologist. The object oriented features of Java allow development of alternative versions of the same coding classes for use by different specialties supporting hierarchical levels of detail and summary or translation of codes.

Display Format

The same information content of a CBPR can be displayed in multiple forms and formats. The dynamic nature of Java makes it particularly suitable for developing interactive custom displays of clinical data. The failure of physicians to recognize abnormal findings in medical records is well

documented. The improvements in legibility and organization which are inherent in a CBPR are often accompanied by increase in data volume and more detailed documentation of normal or redundant findings. Java has now become synonymous with a "wake-up" call for the network. The ease of implementing animation in Java and the ability to use multi-threading to introduce spontaneous and random behavior has enormous potential for increasing the attention given to findings in CBPRs. Java can be used to make reading a CBPR fun and interesting.

Security and Confidentiality

Security and confidentiality controls are key essentials for any CBPR and the subject of much current debate.⁸ Protection of the integrity of the data and controls on access to the data are easy to implement in Java. The Java security package and API is still under design and development, but will eventually provide a standard toolkit for security on the Internet. In the interim, user written Java classes can implement needed encryption, authorization, and authentication routines.

Data Capture Methods

Capturing clinical data remains one of the most important barriers to the growth and dissemination of the CBPR. Pen based computing,⁹ handwriting recognition, and voice recognition are making important advances in facilitating rapid and acceptable physician computer interaction. Unfortunately, Java has little to contribute to integration of new input technologies, but Java is very useful for rapid prototyping of Graphical User Interfaces (GUI) and keyboard input systems.

The main contribution of Java to data capture will probably lie in facilitating complex and high performance user interfaces on low cost machines designed to browse the Internet. Java originated from a consumer electronics project at Sun Microsystems and was originally planned for use on Personal Data Assistants (PDAs). Proposed Java boxes using a microprocessor implementation of the Java virtual machine have much in common with X-window terminals as a user interface device. They are likely to be cheaper, more flexible, and more powerful due to mass marketing potential as Internet browsers. Low cost Java terminals may open up new opportunities for bedside and exam room input and display terminals that may become significant factors in dissemination of CBPR, and offer alternatives to portable computer devices such as pen based computers. Java enables these dedicated Internet access computers to be diskless workstations, but addition of a patient data card interface conforming to ASTM 31.18 standards would expand their

functionality as a clinical data input device.

Clinical Reminders and Automatic Quality Assurance Protocols

The coupling of a CBPR with expert systems to support clinical decision making and automated quality assurance protocols is a key advantage to using a CBPR.¹⁰ Java threads can be used to implement reminder systems which provide immediate feedback during data entry or review. Java can be used to share protocols over the Internet thus encouraging development of libraries of protocols which can be coupled to a greater variety of CBPR implementations. The ASTM has proposed using the Arden Syntax as a basis of writing clinical reminders and Java will be a good medium for making this functionality widely available.

Flow Charts and Graphical Displays

Java is particularly well suited to adding tabular and graphical summary displays to a CBPR in a multi-platform environment. The AWT implements the necessary drawing and windowing tools and helps the programmer to write code which can adapt displays to different print or display capabilities. The use of multiple threads can help to support data exploration and concurrent generation of alternative displays of the same data. Animation can be used to help highlight and illustrate significant trends in data or compare patterns of events over time in different episodes of illness or different patients.

Family Oriented Records

Java threads can help to assemble family oriented records by integrating data from multiple records. Separate Java threads can build and maintain Family problem lists and family encounter summaries for review when any family member is seeing a physician.

SCENARIOS OF CBPR ON THE INTERNET

It is important to consider alternative strategies for implementing a CBPR on the Internet using the Java language.

File Transfer

The simplest approach to transferring CBPR on the Internet is to transfer the entire file in a standardized format such as the ASTM 1238-94 which is independent of the system on which it was created or on which it will be used. This is analogous to the manual methods of photocopying and then mailing or faxing records between physicians subject to written consent from the patient. Java can be used to extract data from the ASTM format and load the data into the local database which will most likely be an SQL server. Any Java-based CBPR should the

ability to import and export patient data in ASTM format. Java classes for this purpose can be shared among developers of Java CBPR. The disadvantage of the file transfer approach is that large amounts of data may be transferred when only a few key entries are needed. Furthermore, file transfer is a one time event and data must be updated in the future.

Java Applets on a Central Server

Java Applets provide a very useful strategy for sharing CBPR over the Internet in a secure and efficient manner. Applets are distributed applications which are attached to HTML web pages and are invoked by an Applet HTML tag. Applets execute on the user's client computer, but may access only data from the same server from which they were launched. Applets provide a safer and more powerful tool than using standard web pages to provide remote access to a CBPR but they lack the full flexibility of Java applications which can access local file systems and have full network access privileges.

Applets can be used to handle encryption and secure data display by encrypting the data before it is stored on the server files and then decrypting it using locally entered passwords and algorithms running on the local client. The main use for Applets will be distribution and exploration of new data and methods from insecure servers. Applets could support some data retrieval activities such as remote access to hospital records or community wide immunization systems. A new category of trusted Applets is under consideration and will probably be added to the Java language

A Patient Record Transport Protocol (PRTTP)

The Hot Java web browser was implemented as a modular extensible application which could be used to add new network protocols easily in the future without replacing the entire program. Current web browsers typically implement Hypertext Transport Protocol (HTTP), File Transfer Protocol (FTP), Gopher, Telnet, and others. One strategy for sharing patient records on the Internet would be to create a new network protocol specifically for using CBPR on the Internet which would include encryption, confidentiality protection, self defining records, and interactive code dictionary transfers. The Java language could implement such a protocol and facilitate adding it to a modular browser such as HotJava which is written in Java.

Client Server Applications Written in Java

Benefits of client-server approaches to CBPR have been demonstrated but rarely used.¹¹ Java is an ideal programming language for writing client server applications which add new functions and services to the Internet. The server program runs on the patient

record server and offers its Internet service on a designated server port number similar to current ftp, telnet, popmail, database and other server protocols. A separate client program, running on the same or a different machine, opens a connection to that port and exchanges data packets through a structured dialog. The server application can also open other needed connections to database servers to obtain data which it needs to satisfy a client request for data.

Using the multithreaded features of Java, a client server CBPR could simultaneously communicate with several different physicians who see that same patient and dynamically build a truly patient centered CBPR by merging data such as problems and medications from the CBPR of the individual physicians.

EXPERIENCE WITH JAVA PROTOTYPES

To test the theoretical advantages and capabilities of Java, we are developing two prototype CBPRs written in Java. The first is a simple ambulatory record designed for teaching CBPR concepts to medical students and residents during their rotations in physician's offices. The Java application communicates over a TCPIP socket with a MUMPS application which is used to store and retrieve the medical record data. Using MUMPS for the data server provides a flexible environment for modeling and exploring a P RTP based on exchange of simple text messages similar in function to HL7 messages. The Java application provides a GUI not available in older MUMPS-based systems. Developing a prototype in Java permits use from multiple sites and a variety of computer platforms. Pilot testing the prototype with medical students facilitates experimentation without the need for concern about production needs of a practice or the need for a complete record. Since students and residents will use only a few records per day, the portability and efficiency of the Data Entry Interface is not a limitation to exploring other features of the CBPR.

The second project uses a Java display client to generate laboratory data summaries from hospital laboratory data stored in an SQL database clinical data repository. We are using a Java Applet for the display client which communicates with the server using a Java controlled TCPIP socket with its own simple security protocol. A Java application is used as a middle-tier information server which uses JDBC to access a Sybase SQL server via an ODBC driver. This approach is similar to the W3-EMR architecture used by Kohane¹², but it is implemented

integrate results from outside laboratories mandated by managed care insurance companies with in-house laboratory results to produce a single summary record.

CONCLUSION

The Java programming language has many features which make it ideal for development of a CBPR in a network environment and for sharing CBPR data between different medical practices over the Internet.

References

1. Dick RS, Steen EB. The Computer-Based Patient Record: An Essential Technology for Health Care. National Academy Press 1991.
2. Cimino JJ, Socratous SA, Claton PD. Internet as Clinical Information System: Application Development Using the World Wide Web. JAMIA 1995;2(5):273-284.
3. Gosling J, McGilton. The Java Language Environment: A White Paper. Sun Microsystems 1995.
4. Hamilton G, Cattell R. JDBC: A Java SQL API. JavaSoft 1996.
5. Greenes R, et. al. Recording, Retrieval, and Review of Medical Data by Physician-Computer Interaction. NEJM 1970; 282:307.
6. Shortlife EH, Perreault LE. Medical Informatics: Computer Applications in Health Care. Addison-Wesley 1990.
7. Lindberg DAB, Humphreys BL, McCray AT. The Unified Medical Language System. Meth Inform Med 1993; 32:281-91.
8. Woodward B. The Computer Based Patient Record and Confidentiality. NEJM 1995; 333(21):1419-22.
9. Poon AD, Fagan LM. PEN-Ivory: The Design and Evaluation of a Ped-Based Computer System for Structured Data Entry. SCAMC 1994; 18:447-51.
10. Barnett GO, Winickoff RN, Dorsey JL, et. al. Quality Assurance Through Automated Feedback Using a Computer-Based Medical Information System. Med Care 1978;16:962-970.
11. Chueh HC, Barnett GO. Client-Server, Distributed Databases Strategies in a Healthcare Record System for a Homeless Population. SCAMC 1993;17:119-124.
12. Kohane IS, Greenspun P, Fackler J, et. Al. Building National Electronic Medical Record Systems via the World Wide Web. JAMIA 1996;3:191-207.

entirely in Java. An extended goal of the system is to